90528 CGAR 2021/22

Computer Graphics - Homework 390528 CGAR 2021/22

Computer Graphics - Homework 2

==========

Ray Tracing: Triangle Meshes and AABB Trees

The goal of this assignment is to implement ray tracing for a triangle mesh, and implement acceleration structures to make the computation faster.

Using Eigen

In all exercises you will need to do operations with vectors and matrices. To simplify the code, you will use Eigen. Have a look at the Getting Started page of Eigen as well as the Quick Reference page for a reference of the basic matrix operations supported.

Preparing the Environment and Submission

Follow the instructions on the *General instructions* document to set up what you need for the assignment.

Ex.1: Triangle Mesh Ray Tracing

In this exercise you will implement the ray-tracing of a triangle mesh. Without an acceleration structure such as a BVH, it will be very slow, so make sure you compile in release mode (`cmake -DCMAKE_BUILD_TYPE=Release ..`), otherwise things will be very slow.

Tasks

1. Fill the starting code to complete the function implementing the intersection of a ray and a triangle. This should be similar to the code you developed for Assignment 2.
2. Fill the starting code to implement the ray-tracing of a triangle mesh.

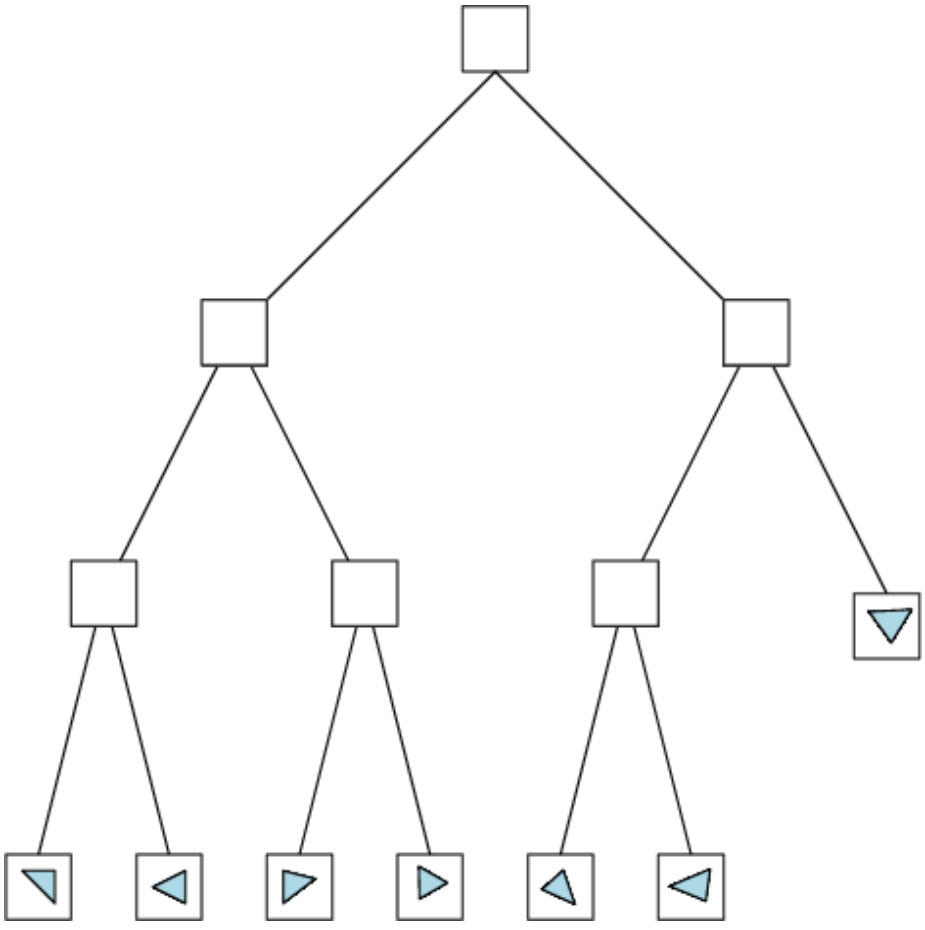
Ex.2: AABB Trees

In this exercise you will implement an acceleration structure to speed-up the rendering of an image via ray-tracing. When compiling in release mode, you will

typically achieve a speedup of 200x to 500x compared to the version without any acceleration structure.

The data structure we will study in this exercise in the AABB Tree, which is a special type of Bounding Volume Hierarchy (BVH), where each node of the tree is associated to the bounding box of its content. More specifically, each leaf nodes contains a single triangle, and stores a box which is the bounding box of this single triangle. Each internal node will have two children (for the sake of simplicity), and stores a box which corresponds to the union of the two boxes stored in its child nodes.

We propose to implement two different approaches to build an AABB Tree for triangles. It is sufficient that you impolement just one of them, but we recommend to try them both.



Top-Down Construction

In this approach, the input triangles are split into groups of roughly equal size starting from the top node. This can be described as a recursive function that does the following:

1. Split the input set of triangles into two sets S1 and S2.
2. Recursively build the subtree T1 corresponding to S1.
3. Recursively build the subtree T2 corresponding to S2.
4. Update the box of the current node by merging boxes of the root of T1 and T2.
5. Return the new root node R.

Sorting Criteria We need a criteria to split the set of input triangles S into two subsets S1 and S2. We propose to simply sort the input triangles based on the coordinate of their centroid along the longest axis of the box spanned by those centroids. Then, S1 will hold the left half (rounded up), and S2 will hold the right half (rounded down).

Bottom-Up Construction

In this approach, we seek to pair nodes of the tree iteratively until only one remains: the root of the tree. Efficient methods to build a BVH with a bottom-up approach are more difficult to implement, since the require intelligent spatial sorting of the input data. We will settle for simple quadratic algorithm, where at each step, we will merge two nodes based on a certain criteria. This procedure can be summarized as follows:

1. Create a leaf node for every input triangle: N1, N2, ... Nk. Let S = {N1, N2, ..., Nk}.
2. Merge two nodes Ni and Nj that minimize some criteria f(Ni, Nj).
3. Update S accordingly (remove Ni, Nj from S, and add merge(Ni, Nj) to S).
4. Repeat until |S| == 1.

Cost Function To evaluate which pair of node we should merge first, you can try one of the following criteria:

- Take f(Ni, Nj) as the distance between the centroid of the boxes associated to Ni and Nj.
- Take f(Ni, Nj) as the increase of volume in the new box (i.e. volume of the union minus volume of Ni and volume of Nj).
- Feel free to try and explore other cost functions as well.

Tasks

1. Implement the intersection test of a ray and an axis-aligned bounding box. This test should be very simple. In particular, there should be no need to solve any linear system here.

2. Implement the top-down or the bottom-up construction method described above.
3. Update the intersection code of a ray and a mesh to use this newly created BVH. In particular, now you should only need to test the intersection for leaf nodes whose bounding box also intersects the input ray.

Starting Code

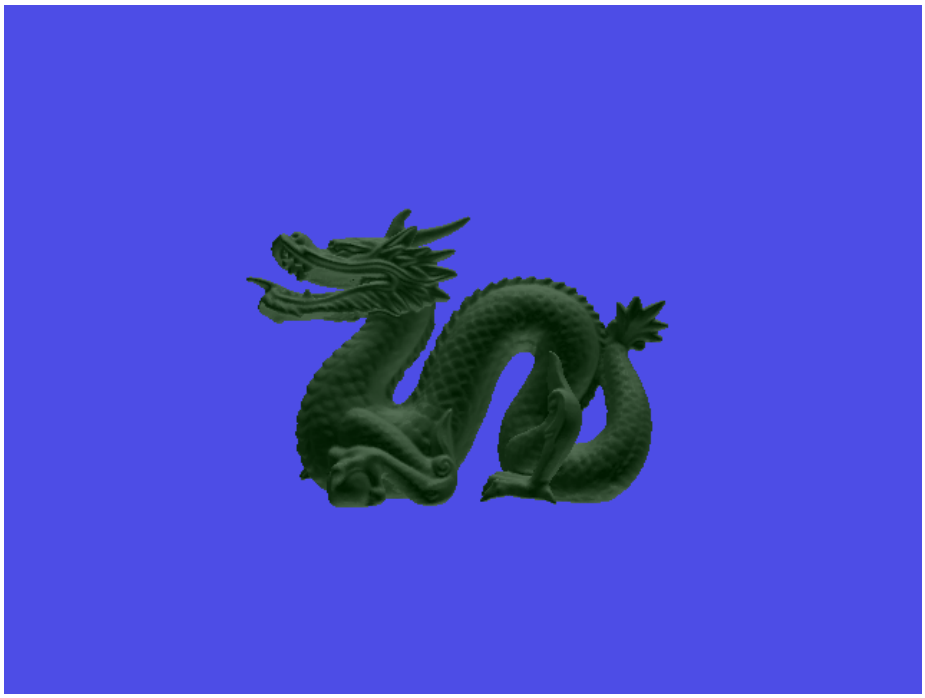
After compiling the code following the process described in the *General instructions* document, you can launch the program from command-line as follows:

```
mkdir build; cd build; cmake ..; make
./assignment4 ../data/scene.json
```

Once you complete the assignment, you should see result a picture generated in your folder.

Running the completed code on the provided file `scene.json` should give you the following result:

Once you have implemented an acceleration structure, you should be able to render more complex objects in a few seconds, such as this [dragon][1]:



NB: To visualize the input 3D models, it is suggested to use a software such as Meshlab.

References

If you are interested in reading more about other acceleration structures, you can have a look at the following references:

- Wald, Ingo, and Vlastimil Havran. *On building fast kd-trees for ray tracing, and on doing that in O (N log N).* 2006 IEEE Symposium on Interactive Ray Tracing. https://doi.org/10.1109/RT.2006.280216
- Samet, Hanan. *Foundations of multidimensional and metric data structures.* Morgan Kaufmann, 2006. ISBN-13 978-0123694461